논문스터디 1주차 1팀

고경현 이진모 장이준

An Introduction to Kernel-Based Learning Algorithms

K. . -R. Muller, S. Mika, G. Ratsch, K. Tsuda and B. Scholkopf, "An introduction to kernel-based learning algorithms," in *IEEE Transactions on Neural Networks*, vol. 12, no. 2, pp. 181-201, March 2001, doi: 10.1109/72.914517.

들어가며,,,

Kernel Method

데이터나 모델의 특성 때문에 기존 데이터의 input space X를 feature space $\mathcal{F}$에 매핑(Φ)함으로써 문제를 해결하는 것이 훨씬 간편한 경우가 있다. 주로 $\mathcal{F}$ 에 속한 데이터들의 내적값을 구할 때가 많은데, 매핑 함수 Φ를 찾는 것이 굉장히 어려운 문제다. 이때 Kernel 함수 K를 이용한다면 $\mathcal{F}$ 에 속한 데이터(벡터)들의 inner product를 Φ를 찾지 않고도 쉽게 구할 수 있다.

식으로 표현하면 아래와 같다.

$$\Phi : X \rightarrow \mathcal{F}$$
$$(x_i,\ x_j) \rightarrow (\Phi(x_i),\ \Phi(x_j))$$
$$< \Phi(x_i),\ \Phi(x_j) >= K(< x_i,\ x_j >)$$

이 방법을 kernel method, kernel trick이라고 부른다. Kernel 방법은 지도 학습과 비지도 학습 기반 방법론에 모두 쓰인다.

본 논문에서는 SVM과 KFD, 그리고 Kernel PCA 방법을 통해 여러 지도 학습과 비지도 학습 과정에서 kernel trick이 쓰이는 수식적 기반과 그 유용성을 검증한다.

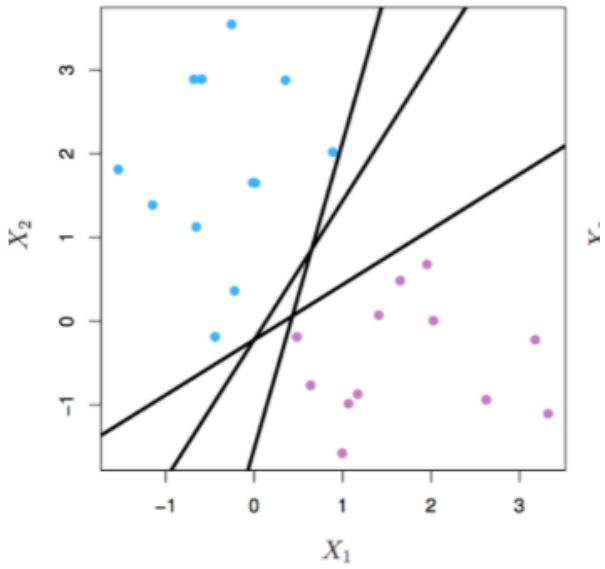
Support Vector Machine (SVM)

SVM을 시작해보자! SVM이 수행하는 작업은 분류다. 데이터를 가장 잘 분류하는 hyperplane을 찾는 것이다.

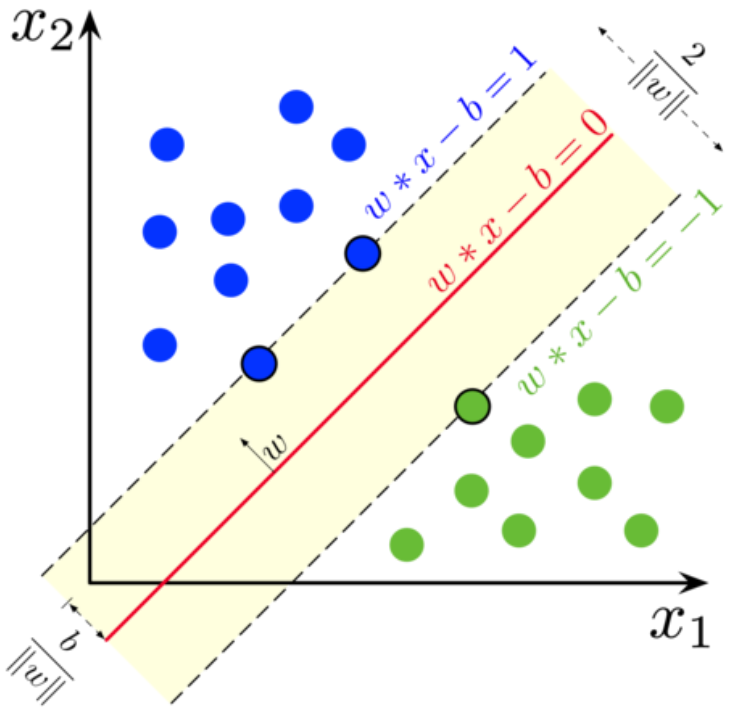
이해를 돕기 위해 초반부는 논문과 약간 다르게 데이터마이닝 수업 내용과 비슷한 흐름으로 진행되며, 수학적인 디테일이 추가되었다고 생각하면 좋다.

I. Model

1. Hard margin & Linear



위와 같은 2차원 평면 위에 두 클래스의 데이터가 있다. 이때 두 클래스가 선형 분류기에 의해서 완벽히 분리되는 separable한 경우라고 가정한다. (= hard margin을 가짐) 만약 이와 같이 가정한다면 위처럼 두 클래스를 분류하는 선형 모델은 검은 선처럼 무수히 많이 존재할 것이다. 그렇다면 과연 이 모델(선)들 중 최적의 모델을 어떻게 찾을 수 있을까? 직관적으로 생각한다면, 두 클래스가 구분되는 지점에서 모델이 한쪽으로 치우쳐 있는 것이 아니라 정중앙에 위치한다면 새로운 데이터를 더 잘 분류할 수 있을 것이기에 최적의 모델이 된다. 최적의 모델을 찾기 위해 margin에 대한 개념이 등장한다.



빨간색 선이 우리가 찾고자 하는 모델 hyperplane이다. margin은 hyperplane에서 가장 가까운 데이터까지의 거리의 두 배이다. 최적의 모델은 이 margin을 최대화하는 모델이며, 해의 유일성을 위해 한 가지 제약 조건이 필요하다.

$$w^T x_i - b \geq +1 \quad \text{for} \;\; y_i = +1 \quad \& \quad w^T x_i - b \leq -1 \quad \text{for} \;\; y_i = -1$$
$$\Leftrightarrow y_i(w^T x_i - b) \geq +1$$

즉 모든 데이터는 margin 바깥에 있어야 한다는 것이다. 이러한 제약 조건 하에 우리가 최대화 하고자 하는 margin M은 다음과 구할 수 있다.

$$w^T x_0 - b = 0 \;\; \text{and} \;\; w^T x_1 - b = 1$$

라고 가정하면, w는 hyperplane의 법선벡터(normal vector)이므로

$$x_1 = x_0 + pw \;\; (p > 0)$$
$$p = \frac{1}{w^T w}$$
$$\therefore M = 2 \cdot |x_0 - x_1| = 2 \cdot |pw| = \frac{2}{||w||_2}$$

가 성립한다. 우리는 이 margin M을 위의 제약 조건 하에 최대화해야 하므로, 역수를 취해 제곱해주면 최소화 문제와 동일해지며 다음과 같이 식을 세울 수 있다.

$$\min_{w,b} \; \frac{1}{2}||w||_2^2 \quad \text{subject to} \;\; y_i(w^T x_i - b) \geq +1$$

위 제약이 있는 최적화 문제를 라그랑지안 승수법을 이용해 제약이 없는 최적화 문제로 바꿀 수 있다.

※ 데마 수업에서 본 식과 약간 다르다. $\beta^T\beta$ = 1 이라는 조건이 없는데, 이는 margin을 계산할 때 positive/negative plane에서 볼 수 있듯이 Support vector들이 속하는 평면의 상수항 값을 1로 제한했기 때문이다.

- Lagrange primal

$$L_P = \frac{1}{2}||w||^2 - \sum_i \alpha_i(y_i(w^T x_i - b) - 1) \quad \text{subject to } \alpha_i \geq 0$$

L_P를 최소화하는 것은 quadratic programming이므로 w와 b로 편미분 해준 값 = 0 으로 두고 방정식을 풀면 α에 대한 식을 얻을 수 있다.

$$\frac{\partial L_P}{\partial w} = w - \sum_i \alpha_i y_i x_i \qquad \frac{\partial L_P}{\partial b} = \sum_i \alpha_i y_i$$
$$\Leftrightarrow w = \sum_i \alpha_i y_i x_i \qquad \sum_i \alpha_i y_i = 0$$

이를 이용해 L_p에 대입하여 primal problem을 dual problem으로 변환가능하다. primal problem을 푸는 것보다 dual problem을 푸는 것이 더 쉬우므로 dual problem으로 변환해서 푼다.

하지만 primal problem의 최적값(해를 대입했을 때 목적함수의 값)는 일반적으로 dual problem의 최적값보다 크거나 같다. 즉, dual problem의 해는 primal problem의 최적값의 하한선만을 보장한다. 하지만 Primal problem의 목적함수와 제약식이 특정 조건 (Slater's condition)을 만족하면 두 문제의 최적값이 동일한 결과를 의미하게 되고, 이때 strong duality를 지닌다고 한다.

- Slater's condition
  1. Primal problem의 목적함수가 convex하고, 부등식 제약조건이 convex, 등식 제약조건이 affine이다.
  2. 최소한 하나의 strictly feasible solution (= 제약조건을 강력하게 만족하는 해)이 존재한다.

따라서 Primal problem은 Slater's condition을 만족하므로 dual problem으로 접근하여 해를 구할 수 있게 된다.

- Dual problem으로의 변환 과정

$$\begin{aligned}\frac{1}{2}\|w\|^2 &= \frac{1}{2}w^Tw \\ &= \frac{1}{2}w^T\sum_{j=1}^{n}\alpha_j y_j x_j \\ &= \frac{1}{2}\sum_{j=1}^{n}\alpha_j y_j (w^T x_j) \\ &= \frac{1}{2}\sum_{j=1}^{n}\alpha_j y_j (\sum_{i=1}^{n}\alpha_i y_i x_i^T x_j) \\ &= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j\end{aligned}$$

$$\begin{aligned}-\sum_{i=1}^{n}\alpha_i(y_i(w^Tx_i-b)-1) &= -\sum_{i=1}^{n}\alpha_i y_i(w^Tx_i-b)+\sum_{i=1}^{n}\alpha_i \\ &= -\sum_{i=1}^{n}\alpha_i y_i w^Tx_i + b\sum_{i=1}^{n}\alpha_i y_i + \sum_{i=1}^{n}\alpha_i \\ &= -\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j + \sum_{i=1}^{n}\alpha_i\end{aligned}$$

- Lagrange dual

$$\therefore L_D = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j$$

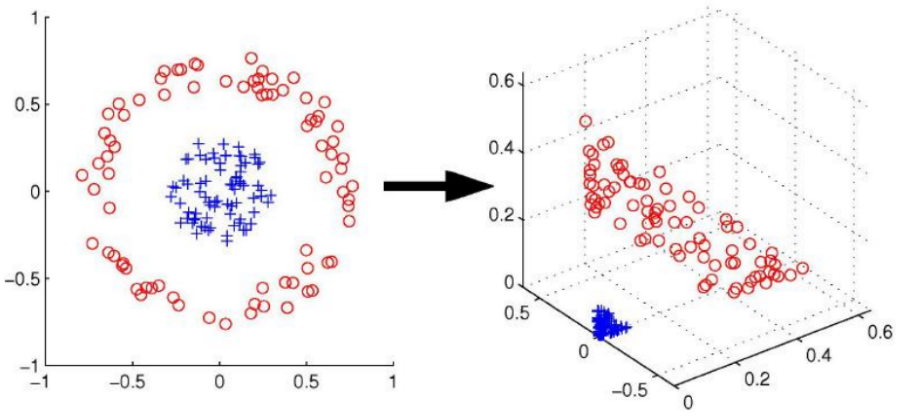
$$\max_{\alpha} L_D \quad \text{subject to} \quad \sum_{i=1}^{n}\alpha_i y_i = 0 \quad \& \quad \alpha_i \geq 0, \quad i = 1, ..., n$$

Primal problem이 최소화하는 문제였으니 Dual problem은 최대화하는 문제가 된다. 위 제약조건을 만족하는 α를 찾는다면 아래 식을 이용해 우리가 원하는 분류 경계면인 초평면(hyperplane)을 찾을 수 있다.

$$f(x) = sgn(\sum_{i}\alpha_i y_i x^T x_i + \ b)$$

2. Hard margin & non-linear

위 경우는 모든 데이터가 변환 없이 하나의 hyperplane으로 완벽히 분류되는 경우에 적용가능하다. 하지만 아래 왼쪽 그림처럼 기존 데이터 공간에서 비선형 분류 경계면이 필요한 경우는 어떻게 해야할까?



(x_1, x_2) $(x_1^2, \sqrt{2}\,x_1x_2, x_2^2)$

이때 바로 Kernel trick이 등장한다. 기존 데이터의 space를 다른 feature space $\mathcal{F}$로 mapping 하여 $\mathcal{F}$에서의 내적값을 쉽게 구해내는 것이 Kernel trick이다.

왼쪽 그림에서 비선형 분류 경계면을 찾는 것은 오른쪽 그림에서 선형 분류 경계면을 찾는 것과 동일한 접근 방법이다. 오른쪽 그림에서 선형 분류 경계면을 찾는 문제는 위 식을 약간 변형하여 아래와 같이 식을 세울 수 있다. 왼쪽에서 오른쪽으로의 mapping을 Φ라고 한다면,

$$\min_{w,b} \ \frac{1}{2}||w||_2^2 \quad \text{subject to} \ \ y_i(w^T\Phi(x_i) - b) \geq +1$$

이를 위와 동일하게 라그랑지안 승수법을 이용해서 Primal problem과 Dual problem으로 나타낼 수 있다. (과정 생략)

- Lagrange primal

$$L_P = \ \frac{1}{2}||w||^2 - \sum_i \alpha_i(y_i(w^T\Phi(x_i) - b) - 1) \quad \text{subject to} \ \ \alpha_i \geq 0$$

- Lagrange dual

$$L_D = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \Phi(x_i)^T\Phi(x_j)$$

$$\text{subject to} \ \sum_{i=1}^{n}\alpha_i y_i = 0 \quad \& \quad \alpha_i \geq 0, \quad i = 1, ..., n$$

식 자체는 1에서 보았던 것과 매우 흡사하다. 다른 점이라면 최적화 문제를 풀기 위해 사용되는 input data가 mapping space $\mathcal{F}$ 상의 데이터 $\Phi(x)$라는 것이다. 우리는 현실적으로 이 mapping Φ를 찾을 수 없기 때문에 mapping된 데이터의 내적값 $\Phi(x_i)^T\Phi(x_j)$을 구하기 위해 Kernel trick이 사용된다.

$$\Phi(x_i)^T\Phi(x_j) = \mathrm{k}(x_i, x_j) \quad \text{, where k kernel function}$$
$$L_D = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathrm{k}(x_i, x_j)$$

따라서 아래 최적화 식을 풂으로써 mapping Φ를 찾는 것 없이 비선형 경계면을 찾을 수 있게 된다.

$$\max_{\alpha} L_D \quad \text{subject to} \quad \sum_{i=1}^{n}\alpha_i y_i = 0 \quad \& \quad \alpha_i \geq 0, \quad i = 1, ..., n$$

3. Soft margin & non-linear

지금까지 살펴본 두 가지 케이스는 모두 데이터가 hard margin을 갖는, 즉 데이터가 선형/비선형 분류 경계면에 의해 완전히 분류되는 경우였다. 하지만 현실에는 noisy한 데이터의 경우가 더 많고, 이를 완화하는 모델이 바로 최종적인 Support vector machine(SVM)이다. SVM은 slack variable을 도입하여 약간의 error ξ를 허용하는 분류 경계면을 찾는 것을 목표로 한다. SVM의 목적함수를 살펴보자.

$$\min_{w,b,\xi} \ \frac{1}{2}||w||_2^2 + C\sum_i \xi_i \quad \text{subject to} \ \ y_i(w^T\Phi(x_i) - b) \geq 1 - \xi_i, \ \ \xi_i \geq 0$$

직전에 보았던 hard margin & non-linear case에서 error를 허용하는 term $C\sum_i \xi_i$가 추가됨으로써 soft margin인 케이스로 바뀌었다. C는 error를 얼마나 허용할 것인지 정하는 regularization constant다.

- Lagrange primal

$$\min L_P = \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i(y_i(w^T\Phi(x_i) - b) - 1 + \xi_i) - \sum_{i=1}^{n}\beta_i\xi_i$$
$$\text{subject to} \quad \alpha_i \geq 0, \quad \beta_i \geq 0, \quad i = 1, ..., n$$

동일하게 w, b, ξ에 대해 편미분 후 미분값 = 0으로 두고 전개해 α에 대한 식을 얻을 수 있고, Lagrange dual problem으로 변환이 가능하다.

$$\frac{\partial L_p}{\partial w} = 0 \quad \rightarrow \quad w = \sum_{i=1}^{n} \alpha_i y_i \Phi(x_i)$$
$$\frac{\partial L_p}{\partial b} = 0 \quad \rightarrow \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$
$$\frac{\partial L_p}{\partial \xi_i} = 0 \quad \rightarrow \quad C - \alpha_i - \beta_i = 0$$

또한 KKT condition을 만족하기 때문에 primal solution과 dual solution은 모두 최적해를 보장한다.

- Lagrange dual

$$\max L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathrm{k}(x_i, x_j)$$
$$\text{subject to} \quad \sum_{i=1}^{n} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \quad i = 1, ..., n$$

앞서 2번 케이스에서 보았던 것과 크게 다르지 않다. 다른 점은 라그랑지안 승수 α에 상한 C가 생겼다는 것이다. C의 존재로 인해 우리 모델은 error를 일정 수준 허용할 수 있는, 조금은 더 robust한 모델이 되었다. 위 최적화 식을 해결함으로써 비선형 분류 경계면을 찾을 수 있다.

II. Attributes

SVM의 특징들을 알아보자

1. Sparsity

KKT condition의 네 가지 조건 중 complementary slackness 조건에 의해 SVM의 soultion은 sparse한 특징을 갖는다. 대부분의 α가 0이라는 것인데 왜 그런지 알아봊.

Complementary slackness 조건은 부등식 제약 조건과 거기에 곱해지는 라그랑지안 승수 둘 중 적어도 하나는 0이어야 한다는 조건이다. 이를 풀어쓰면 아래와 같다.

$$\alpha_i (y_i (w^T \Phi(x_i) - b) - 1 + \xi_i) = 0, \ \beta_i \xi_i = 0, \ \text{for all } i$$

위 식을 만족하는 동시에 다른 조건들

$$y_i (w^T \Phi(x_i) - b) \geq 1 - \xi_i$$
$$\xi_i \geq 0$$
$$C - \alpha_i - \beta_i = 0$$

도 만족시켜야 하므로

α_i의 범위를 나누어서 보면

1) $\alpha_i = 0 \quad \Rightarrow \quad y_i f(x_i) \geq 1 \ \text{ and } \ \xi_i = 0$

2) $0 < \alpha_i < C \quad \Rightarrow \quad y_i f(x_i) = 1 \ \text{ and } \ \xi_i = 0$

3) $\alpha_i = C \quad \Rightarrow \quad y_i f(x_i) \leq 1 \ \text{ and } \ \xi_i \geq 0$

이와 같은 등식/부등식이 성립한다.

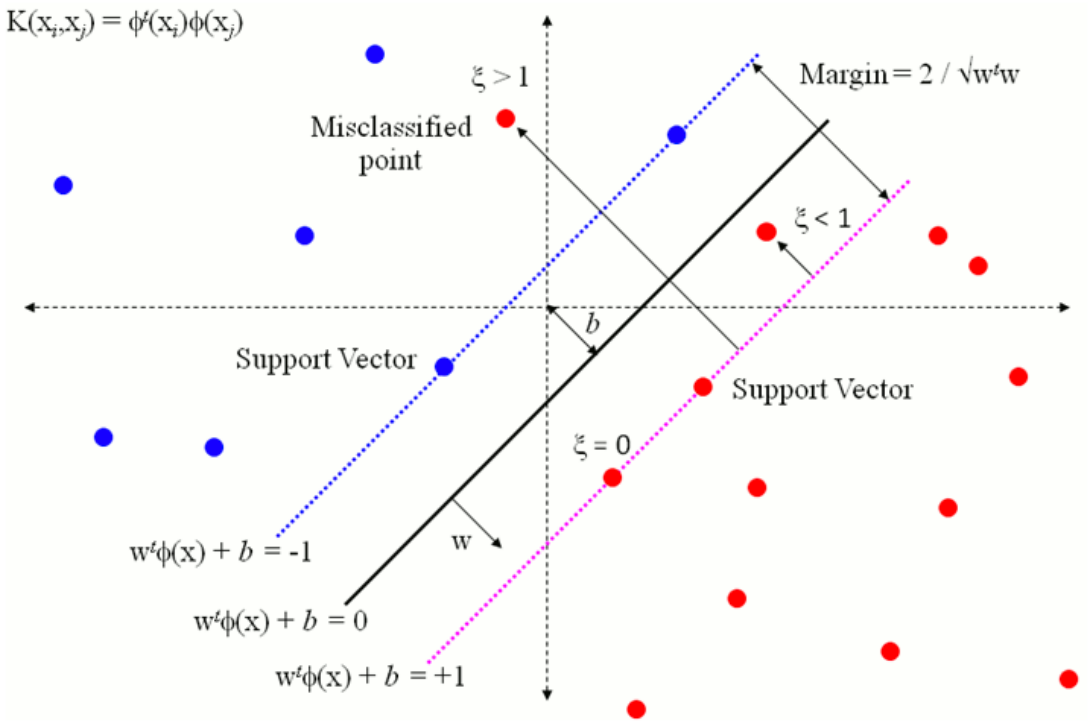
이를 해석해보면

1) α가 0인 데이터들은 올바르게 분류 되어 있고

2) 0과 C사이에 있을 땐, margin의 경계에 있거나

3) C일 땐, margin의 안쪽에 있다는 의미이다.

좀 더 보자.



Support vector(SV)의 의미와 연관지어 설명할 수 있다. SV는 hyperplane을 결정짓는 data point이다. 대부분의 데이터는 올바르게 분류되며, SV만이 분류 경계면을 결정지으므로 α는 대부분이 0인 sparse한 특징을 갖는 것을 알 수 있다. 또한 이러한 sparse한 특징 덕분에 SVM 모델을 학습하는 비교적 적은 시간이 소모된다.

2. ν-SVM

앞선 SVM의 변형된 버전인 ν-SVM이 있다. C대신 새로운 파라미터 ν를 사용하는 것이 특징이다. 목적함수는 아래와 같다.

$$\max -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathrm{k}(x_i, x_j)$$

$$\text{subject to} \quad \sum_{i=1}^{n}\alpha_i y_i = 0, \quad 0 \le \alpha_i \le 1/n, \quad \sum_i \alpha_i \ge \nu, \;\; \nu \in (0, 1]$$

두 번째 제약 조건을 보자. α_i의 상한은 ν의 값과 상관 없이 $1/n$로 고정이다. 그렇다면 $\sum_i \alpha_i$는 절대 1을 넘지 못한다(i.e. $\sum \alpha_i \le 1$). 여기서 만약 ν가 1이라고 한다면 $\sum_i \alpha_i = 1$을 만족시켜야 하고, 이를 만족시키기 위해서 모든 α_i는 $1/n$가 된다. sparsity 부분을 보면 α_i의 값이 상한선일 때 해당하는 데이터(벡터)는 Support vector가 된다. 모든 α_i가 $1/n$이므로 적어도 n개의 벡터가 Support Vector여야 하며, 최소한 n개의 벡터가 margin 내에 있어야 한다.

이를 통해서 ν가 SV의 하한선과 Margin error의 상한선을 결정하는 것을 알 수 있다.

3. Computing b

지금까지 w또는 α를 구하는 방법에 대해서만 배웠다. 하지만 기존 hyperplane 모델은 절편인 b를 포함하고 있다. SVM에서 이를 threshold라고 표현한다. threshold를 계산하는 방법에 대해 알아보자. 앞선 sparsity 설명 중 $0 < \alpha_i < C$ 일 때, $y_i f(x_i) = 1 \;\text{ and }\; \xi_i = 0$가 성립하는 것을 알 수 있었다. 이 중 $y_i f(x_i) = 1$ 식을 이용해 간단하게 b를 계산할 수 있다.

$$y_i(b + \sum_j y_j \alpha_j \mathrm{k}(x_i, x_j)) = 1$$

이고,

$$0 < \alpha_i < C$$

을 만족하게 하는 index i들의 집합을 I라고 할 때, 위 식을 i에 대해 모두 더한 후 평균을 내어 b를 계산할 수 있다. 식은 다음과 같다. ($|I|$는 집합 I의 원소의 개수)

$$b = \frac{1}{|I|}\sum_i (y_i - \sum_j y_j \alpha_j \mathrm{k}(x_i, x_j))$$

closed form을 통해 수학적으로 굉장히 안정적인 (stable) 결과를 도출한다.

4. Geometrical explanation

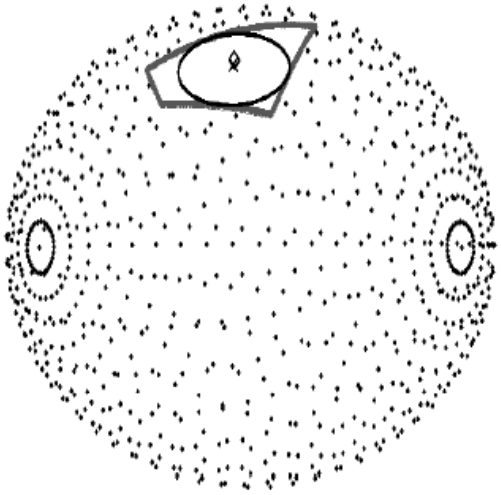


Fig. 5. An example of the version space where the SVM works fine. The center of mass (◇) is close to the SVM solution (×). Figure taken from [72].

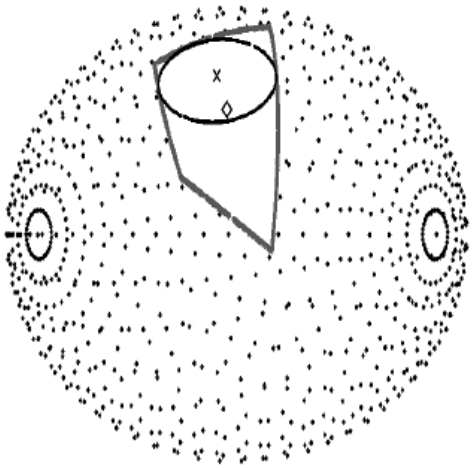


Fig. 6. An example of the version space where SVM works poorly. The version space has an elongated shape and the center of mass (◇) is far from the SVM solution (×). Figure taken from [72].

SVM의 solution을 기하학적인 관점에서 바라볼 수 있다. 그림의 수많은 점들은 해를 구하기 위한 다양한 가설들이고, 그 중 회색 두꺼운 선으로 표시된 공간은 후보 모델이 될 수 있는 가설들의 집합이 이루는 공간이다. 이때 SVM의 solution은 이 가설 공간 내의 원을 최대로 만드는 중심 ×에 위치한다.

그리고 ◇은 Bayes solution인데, 위 두 그림처럼 SVM solution은 Bayes solution과 일치할 수도 있고, 아닐 수도 있다.

아래 개념들을 이용해 설명할 수 있다.

논문에서 SVM의 solution은 **Version space $\mathcal{V}$ 의 체비셰프 중심**과 일치한다고 한다.

두 개념에 대한 설명은 다음과 같다.

- **체비셰프 중심 (Chebycheff center)**

  주어진 Set 내의 hypersphere의 크기 최대로 만드는 중심 벡터.

- **Version space**

$$\mathcal{V} = \{h : h \in \mathcal{H}, \text{consistent}(h, D)\}$$

  모든 가설을 포함하는 가설 공간 $\mathcal{H}$에서 consistent한 가설 h들만을 모아놓은 집합 $\mathcal{V}$

  - consistent

$$\text{consistent}(h, D), \text{ if } h(x) = c(x) \text{ for } x \in D, c \text{ is target concept}$$

    D는 training examples의 집합일 때, 가설 h와 집합 D가 consistent하다는 것은, 가설 h가 아래와 같은 조건을 만족함을 의미한다.

라고 하는데 솔직히 와닿지가 않는다.. 이를 우리가 익숙한 용어로 설명해보겠다.

수많은 모델들의 집합 $\mathcal{H}$에서 우리가 원하는 모델들의 집합이 바로 $\mathcal{V}$라는 것이고, $\mathcal{V}$에 속하는 모델들은 train data set D을 타겟 모델 c에 넣었을 때 같은 (consistent한) 결과를 도출한다.

III. Optimization

SVM의 solution은

$$\max L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathrm{k}(x_i, x_j)$$
$$\text{subject to} \quad \sum_{i=1}^{n} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \quad i = 1, ..., n$$

을 해결함으로써 풀 수 있다. 위는 $-\frac{1}{2}\alpha^T K \alpha + \mathbf{1}^T \alpha$ 꼴이니까 Convex quadratic programming(QP) 문제다. (QP는 목적함수가 이차식 형태이며 matrix가 psd인 최적화 문제를 말함)

그러면 QP를 해결하는 코드(파이썬, R 등등..)를 이용해 해를 찾으면 될까? 그렇다면 참 좋겠지만 standard QP를 해결하는 알고리즘들은 quadratic term(여기서는 K)이 sparse한 경우에 잘 작

동한다. 하지만 SVM의 K는 dense하기에 optimization에 굉장히 많은 비용이 소모된다. 그래서 최적값에 비교적 빠르게 수렴하는 세 가지 최적화 방법을 소개하겠다.

문제는 sparse하지 않은 K 때문에 수렴 속도가 늦어지는 것이다. 그러면 수렴 속도를 빠르게 하기 위해 필요 없는 element들을 쳐내어 K의 사이즈를 줄이면 되지 않을까? 이러한 아이디어를 이용하여 아래 세 가지 방법은 모두 하나의 QP를 여러개로 쪼개어 문제를 푸는 방법들이다.

1. Chunking

Chunking은 soultion의 sparse한 특징과 KKT condition을 이용한 방법이다. 하나씩 보자

1. (Sparsity) α_i가 0이라면 matrix K의 i번째 행과 i번째 열의 값은 필요가 없어진다. (계산할 때 0이랑 곱해져서 0이 되니까)
2. (KKT condition) α_i가 KKT condition을 만족해야 optimal하다.

   dual solution(α)이 KKT condition을 만족 → dual solution과 primal solution이 동일한 결과

   만약 solution α가 KKT condition을 만족시키지 않는다면, primal solution과 같다고 보장하지 못하므로 optimal solution이 아니게 된다.

따라서 위 두 가지 조건을 모두 만족시키는 α는 0이기 때문에 계산이 무의미해지고, optimal하기 때문에 계산에서 제외될 수 있다. 다시 말해서, 0이 아닌 α와 KKT condition을 만족시키지 않는 α만 가지고 최적화 식을 구성할 수 있게 되고, 사이즈가 줄어든 K덕분에 수렴 속도가 증가하게 된다.

2. Decomposition

Chunking과 비슷하게 작은 size의 matrix를 만들어 최적화를 빠르게 하는 방법이다. Chunking과 다른 점이라면 size가 처음부터 고정된 채 알고리즘이 동작한다는 것이다.

Index set I를 두 그룹 B, N으로 분류한 후 한 그룹에 속하는 데이터에 대한 α값은 고정시킨 채 나머지 그룹에 속하는 데이터로만 최적화를 하는 것이다. 이때 적어도 하나의 데이터가 KKT condition을 만족시키지 않는다는 것이 특징이며, 고정된 사이즈와 캐시를 사용하기 때문에 빠른 수렴 속도를 보인다고 한다.

3. Sequential Minimal Optimization (SMO)

SMO는 decomposition의 극단적인 경우이다. decomposition은 하나의 QP를 두 개로 분리시켰다면 SMO는 QP를 더 잘게 쪼개어 수렴 속도를 증가시킨다. 최소한으로 쪼갤 수 있는 경우는 데이터 2개 (=라그랑지안 승수 2개)만 포함하는 경우이다(왜냐면 선형 제약 조건을 만족시켜야 하기 때문에). 여기서 SMO의 특징이 드러나는데, 데이터가 2개 밖에 없는 채 최적화를 하다보니 기존의 QP 알고리즘이 전혀 필요 없어진다. (그저 방정식만 풀면 되는 듯?, 논문에서는 analytically하게 이루어진다고 한다.) 따라서 굉장히 빠른 속도를 보이며, Support vector regression과 one-class SVM에서도 최적화 시 SMO를 사용한다.

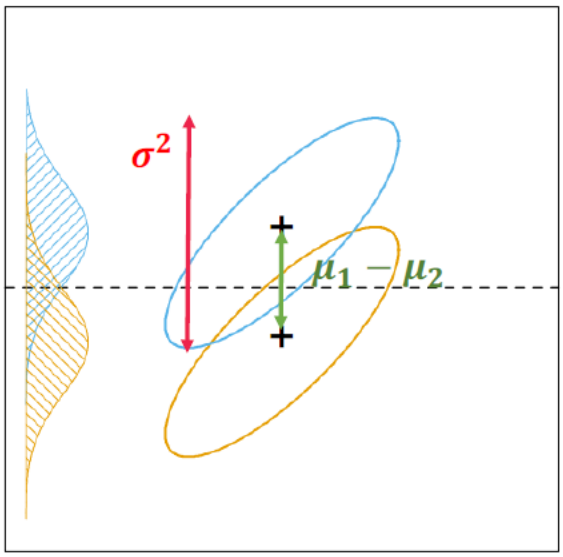
여기 등장하는 세 가지 Optimization은 이 논문(”Fast Training of Support Vector Machines Using Sequential Minimal Optimization”)이 비교적 잘 설명하고 있다.

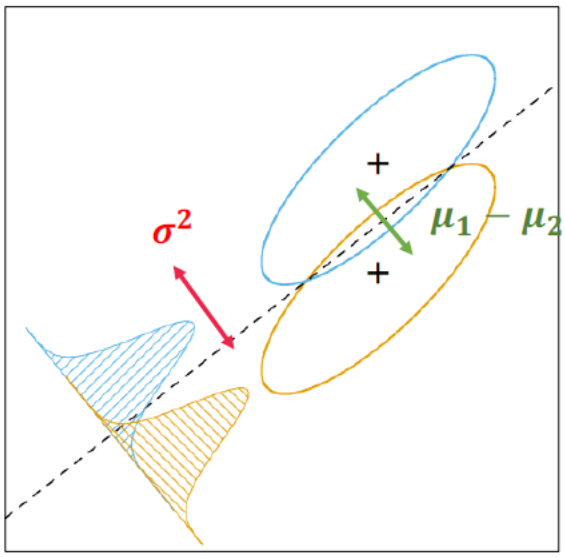
참고

KFD

데이터의 분포를 학습하여 boundary를 만들어주는 Kernel Fisher Discriminant에 대해 본격적으로 알아가보려고 한다. 그 전에 KFD의 기저라고도 할 수 있는 LDA에 대해 먼저 알아보고, LDA에 Kernel 개념을 추가한 KFD를 설명할 예정이다. 추가적으로 kernel method와 boosting은 어떻게 함께 쓰이는 지에 대해서도 설명할 예정이다.

I. LDA: Linear Discriminant Analysis

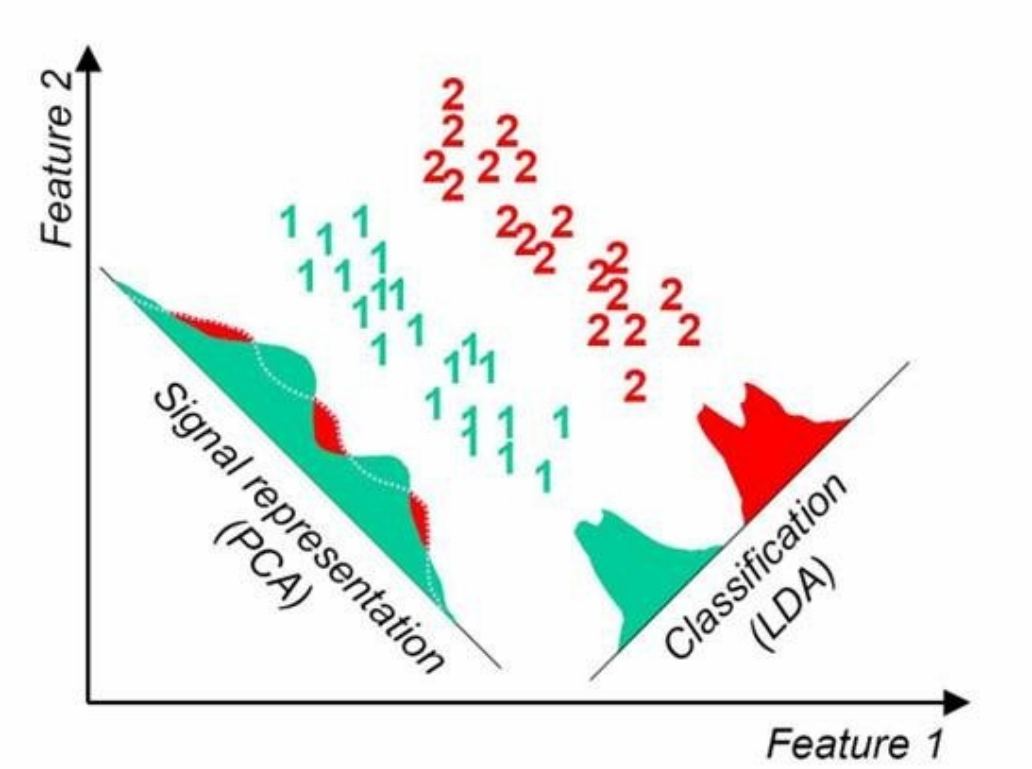




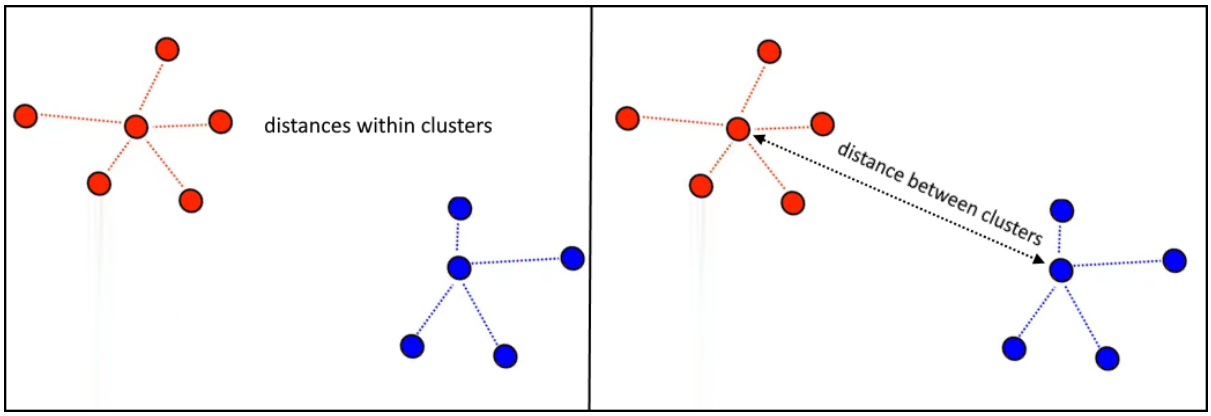
LDA란 데이터가 주어졌을 때 이를 특정 기저에 정사영을 시켜서 데이터를 분류해내는 모델이라고 보면 된다. (LDA하면 토픽모델링 기법인 Latent Dirichlet Allocation가 자주 등장하는데, 이와는 완전히 다른 분류 모델이다.)

LDA는 데이터를 차원축소시키면서 Y값에 대한 분류를 극대화하려는 것이 목표이다. 예를 들어, 왼쪽과 같은 경우는 정사영시킬 시 데이터 간의 분산은 크지만 주황색 데이터와 파란색 데이터 분포 간의 중복된 파트가 큰 경우를 보아 분류가 잘 되지 않았음을 알 수 있다. 반대로 오른쪽과 같은 경우에는 정사영시킨 후의 데이터 간의 분산이 상대적으로 작다고 볼 수 있지만 Y값이 분류되게끔 정사영되어 LDA는 최종적으로 오른쪽과 같은 정사영 기저를 택할 것이다.

정사영시켜 차원 축소한다는 점에서는 PCA와 굉장히 유사한 것 같은데, 과연 어떤 점이 다른 것일까? 이미 눈치챈 분들도 있겠지만, PCA는 정사영시킬 기저를 정할 때 “분산이 가장 큰 쪽으로” 데이터가 분포하게끔 하는 것이 목표이다. 그렇지만 LDA는 projection 이후에도 두 범주가 잘 분류되게끔 하는 것이 목표이다. 그래서 하단의 사진과 같은 차이가 등장하게 되는 것이다.



LDA에 대해 본격적으로 알아가기에 앞서 clustering에서도 몇 번 등장한 Between distance, within distance에 대한 개념을 알아야 한다.



오른쪽 사진에 보이는 것처럼 between distance란 각 범주간 데이터의 중심인 centroid간의 거리를 말한다. (군집 간 거리). within distance란 반대로 각각 객체들과 해당 범주의 centroid간의 거리, 즉 군집 내 거리를 나타낸 것이다.

만약 데이터를 잘 분리를 해내려면 clustering의 objective와 비슷하게 between distance는 최대화, within distance는 최소화하는 방향으로 학습이 이루어져야만 한다.

본격적으로 LDA의 수식에 대해 알아보도록 하겠다. 우리는 Fisher's의 LDA를 기준으로 설명할 것이다.

$$y = w^T x \qquad m_1 = \frac{1}{N_1}\sum_n x_n \qquad m_2 = \frac{1}{N_2}\sum_n x_n$$

y는 x와 w에 내적한 것이고, m1은 첫번째 군집의 centorid, m2는 두번째 군집의 centroid이다.

이때, 첫번째 목적 함수는 projection 이후 Between distance를 최대한 크게 만드는 것과 관련되어 있다.

$$M_2 - M_1 = w^T(m_2 - m1) \Rightarrow M_k = w^T m_k$$

두번째 목적 함수는 projection 이후 Within distance를 최대한 작게 만드는 것과 관련되어져 있다.

$$S_k = \sum_n (y_n - m_k)^2$$

이 두 가지를 모두 고려하여 J(w) 라는 새로운 목적 함수가 탄생하였다.

$$J(w) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} = \frac{W^T S_B W}{W^T S_w W}$$

$$S_B = (m_2 - m_1)(m_2 - m_1)^T$$
$$S_W = \sum_n (x_n - m_1)(x_n - m_1)^T + \sum (x_n - m_2)(x_n - m_2)^T$$

J(w)의 분자에는 군집 간의 거리, 분모에는 군집 내의 거리이고, 군집 간의 거리는 증가시키고 군집 내의 거리는 줄이는 방향으로 학습해야 하므로 J(w)를 최대화시키는 w를 찾아내면 된다.

목적함수인 J(w)를 w로 편미분하게 되면

$$\frac{\partial J(w)}{\partial w} = \frac{(W^T S_B W) S_w W - (W^T S_w W) S_B W}{W^T S_w W * W^T S_w W} = 0$$

꼴이 나오고, 해당 식을 0으로 만드는 w를 찾으면 된다. 이때 분모는 0이 되면 안되므로

$$(W^T S_B W) S_w W - (W^T S_w W) S_B W = 0$$
$$\Downarrow$$
$$(W^T S_B W) S_w W = (W^T S_w W) S_B W$$

가 성립하게 된다. 이를 변형해주면

$$\frac{(W^T S_B W)}{W^T S_w W} S_w W = S_B W$$

이렇게 완성이 되는데, 이때

$$\frac{(W^T S_B W)}{W^T S_w W} = \alpha$$
$$S_B W = (m_2 - m_1)(m_2 - m_1)^T W = \beta(m_2 - m_1)$$

로 각각의 요소들을 부분적으로 **α, β** 라는 scalar형태로 바꿔줄 수 있다.

$$\frac{\alpha}{\beta} W = S_w^{-1}(m_2 - m_1) \Rightarrow W \propto S_w^{-1}(m_2 - m_1)$$

즉 해당 식에 의하여 W의 방향성은 S_w^{-1}(m_2-m_1) 에 의해 정해지는 것이다.

상단의 과정은 KFD에서도 동일하게 적용된다.

II. KFD: Kernel Fisher Discriminant

$$x \rightarrow \Phi(x_i)$$

위의 LDA 과정에서 x를 kernel함수에 집어넣은 것으로 바꿔주고,

$$w = \sum_i \alpha_i \Phi(x_i), \quad \alpha_n \in R$$

projected vector인 w도 Φ(Xn)의 span꼴이라고 가정한다.

이때 projected mean은

$$w^T m_i^\Phi = \frac{1}{N} \sum\sum \alpha_n (\Phi(X_n)\Phi(X_k^i)) = \frac{1}{N} \sum\sum \alpha_n K(X_n, X_k^i) = \alpha^T * M_i$$
$$, K_{ij} = (\Phi(X_i)\Phi(X_j)) = k(x_i, x_j)$$

위 두 가정 하에 LDA와 비슷한 수순의 과정을 거치게 되면 KFD의 objective는 결국 위와 유사한

$$J(\alpha) = \frac{w^T S_B^T w}{w^T S_W^T w} = \frac{\alpha^T M \alpha}{\alpha^T N \alpha}$$
$$, M = (M_2 - M_1)(M_2 - M_1)^T$$
$$, N = \sum K_j (I - 1_N) K_j^T$$

꼴로 등장하게 된다. (위의 증명 과정은 자료로 첨부하겠습니다!) 여기서 w가 아닌 **α**가 왜 등장하냐고 질문이 나올 수도 있는데, 최적화할 w를 Φ(Xn)와 α의 span꼴로 치환해주었기 때문이다. LDA와 비슷하게 α에 관하여

$$\alpha \propto N^{-1}(m_2 - m_1)$$

이라는 식으로 귀결된다. 최종적으로 우리가 구하고자 하는 Y는 하단과 같이 구할 수 있다.

$$y(x) = w * \Phi(X_i) = \sum \alpha_n * K(x_i, x)$$

1. Using KFD

kernel function과 데이터만 주어지면 feature space에 위치한 α값을 구해낼 수 있고, 이를 기반으로 projection을 진행할 수 있다. 정사영 후의 분류과정에서는 적절한 threshold가 필요한데,
1) projection 이후 두 classes간의 평균값

2) 혹은 SVM로 학습하여

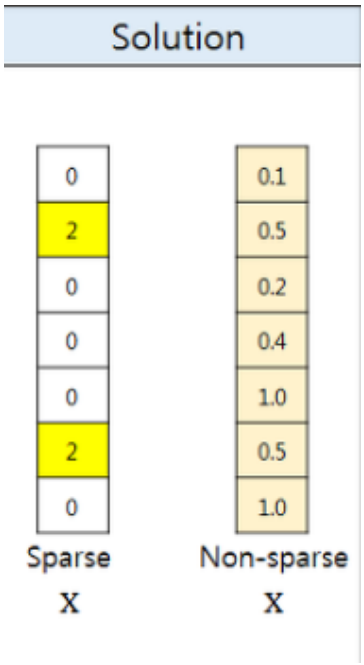
분류를 진행할 수 있다.

2. Regularization

feature space의 dimension은 학습 데이터의 개수보다 같거나 크므로 overfitting이 일어날 가능성이 크다. 그래서 regularization이 필수인데, 이때 α, w에 penalty를 주는 식으로 진행된다. (reference 87, 88을 참고하자)

3. KFD problem: non-sparse solutions

위에서 언급했듯 KFD에서 결국 구해야 하는 것은 Mα=λNα인데, 이를 eigenvector problem으로 해석하여 α=N^-1(m2-m1)으로 정답을 구해낼 수 있다. 다만 이에는 여러 문제점들이 존재하는데, N와 M의 스케일이 training sample에 따라 달라진다는 점. 그리고 solution이 non-sparse하다는 점이 있다. 여기서 non-sparse한 solution에 대한 것이 무엇인지 하단에 첨부하겠다. (간단히 말하자면, zero인 변수가 거의 없는 solution을 말하는 것이다.)



해결하기 위해 제약식을 추가로 두기로 결정하였고, 이후에 등장한 것이 바로 convex quadratic programming problem이다.

4. KFD Solution: Convex Quadratic Programming Problem

-reference 89, 90

KFD를 convex quadratic programming problem로 transform하며 위의 문제점을 해결할 수 있다.

$$J(\alpha) = \frac{\alpha^T M \alpha}{\alpha^T N \alpha}$$

를 최대화시키는 α를 구하는 것이 우리의 목표이고, M은 rank가 1이다.(ex. αMα=(α(m2-m1))^2) 그래서 (α(m2-m1))를 non-zero value로, 예를 들어 2로 고정시킨 다음 αNα을 최소화시킨다.

그렇다면 목적함수가

$$\begin{gathered} min \quad \alpha^T N \alpha + CP(\alpha) \\ subject \quad to \quad \alpha^T (m_2 - m_1) = 2 \end{gathered}$$

이러한 식으로 바뀔 수 있다. 여기서 P는 위에서 언급한 regularizer이고, C는 regularization constant에 해당된다. 그렇지만 N는 여전히 비직관적이기 때문에 α를 최적화시키기에는 부적합하다고 볼 수 있다. 그래서 N이라는 요소를 제외해여 새로운 형태로 변형해주었다.

$$\begin{gathered} min \|\xi\|^2 + CP(\alpha) \\ subject \quad to \quad K\alpha + 1b = y + \xi \quad 4(a) \\ 1_k^T \xi = 0 \quad for \quad k = 1, 2 \quad 4(b) \end{gathered}$$

여기서 4(a) 제약식은 w*x_i + b =y_i + ξ 로 해석할 수 있거, 이는 각각의 sample들을 자신의 class에 연결짓는 것이라고 볼 수 있다. ||ξ ||^2를 최소화시키는 것은 곧 error의 분산을 줄이는 것을 의미하고, 동시에 4(b)는 각 class의 ouput의 평균이 곧 label임을 의미한다.

5. Optimization

convex quadratic programming problem으로 tranfrom하여 효율적인 계산을 할 수 있다.

이때 P(α)를 L1-norm regularizer인 P(α) = ||α||로 바꾸게 되면 sparse한 solution을 얻을 수 있다. → SFKD

그에 대한 이유를 간단히 설명해보자면,

ex) $\|(1,0)\|_2 = \|(1/\sqrt{2}, 1/\sqrt{2})\|_2 = 1$

이지만 $1 = \|(1,0)\| < \|(1/\sqrt{2}, 1/\sqrt{2})\| = \sqrt{2}$

인 것을 보았을 때 L1-norm으로 최적화시키는 것이 (1,0)과 같은 더 sparse한 solution을 도출해냄을 알 수 있다.

더 나아가 ξ을 L1-norm 형태로 바꾸게 되면 직관적으로도 선형적인 문제풀이로 바뀔 수 있음을 알 수 있다. → LSKFD

III. Connection between Boosting and Kernel Methods

-reference 93,53, 94, 95, 96, 97

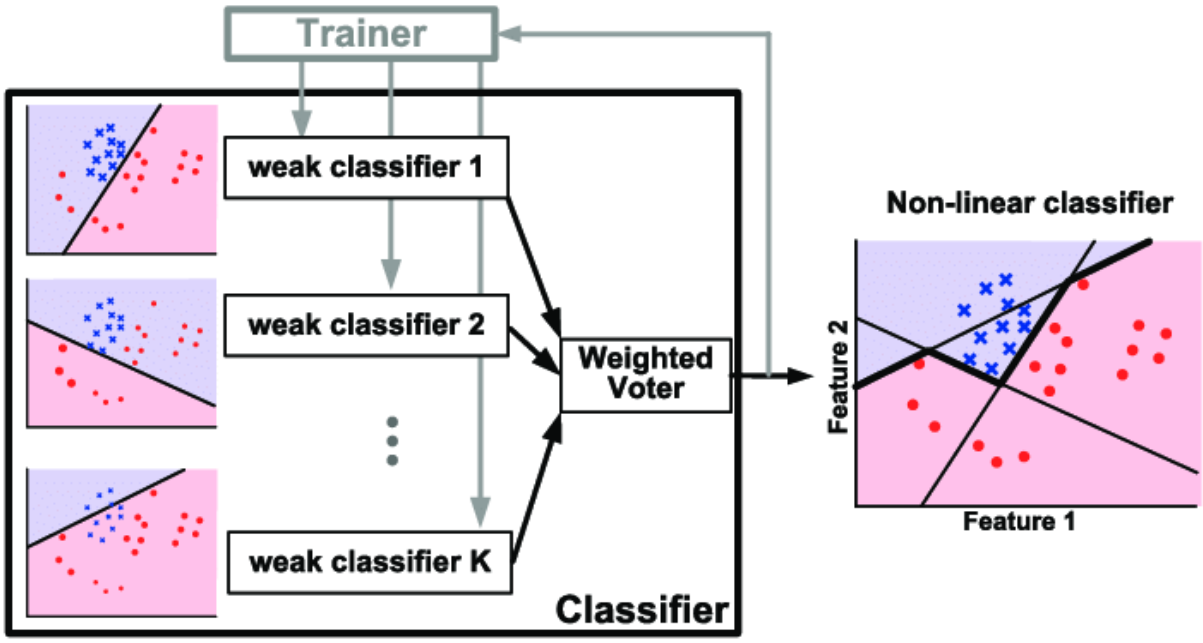
이제 우리는 boosting과 앞에서 보았던 kernel methods들인 SVM, KFD간의 관계를 볼 것이다.

그전에 AdaBoost에 대해 가볍게 살펴볼 필요가 있다.

weak model이란, 정확도가 random하게 guessing하는 것보다 약간 더 좋은 성능을 내는 모델을 말한다. 이를 boost을 하게 되면 좋은 성능을 내는 strong model로 발전시킬 수 있다.

학습 과정을 간단히 몇 줄로 요약하자면, weak model로 우선 학습을 진행하고, 이 모델이 잘 예측하지 못한 sample들에 대해 reweight를 하고, 두번째 weak model로 reweight된 sample을 학습시킨다.

이러한 식으로 학습을 반복한 뒤에 실제 예측 시에는



학습된 weak learner들을 순차적으로 지나며 각 model들의 결과물을 결합하여 예측해낸다.

최종모델인 f는 h라는 convex functions, 즉 weak learner들의 combination으로 이루어져있고

$$f(x) = \sum_t \frac{w_t}{\|w\|} h_t(x)$$

각각의 weak learner들은 각기 다른 분포인 p=[p1,,,pn]로 가정되어져 있다. 이때 p_i라는 분포는 이전모델에 학습이 잘되지 않은 데이터들에 가중치를 부여한 분포라고 보면 된다.

이러한 adaboost를 변형한 것이 바로 arc-gv이다. 이것은 convex combination인 sum(h(x))가 margin을 최대화시킨다는 것을 증명해낸다. (SVM의 hard margin을 구하는 것과 비슷하다.)

안녕하세욜ㅋㄹ

실제 arc-gv의 solution은 margin p를 최대화시키는 linear program과 비슷하다.

$$\begin{gathered} max \quad \rho \\ subject \quad to \quad y_i \sum_j w_j h_j(x_i) \geq \rho \\ for \quad i = 1...n \\ \|w\|_1 = 1 \end{gathered}$$

이때, SVM의 아래와 같은 식이

$$\min_{w,b} \ \frac{1}{2}||w||_2^2 \quad \text{subject to} \ \ y_i(w^T \Phi(x_i) + b) \geq +1$$

하단과 같이 재정의될 수 있다.

$$\begin{gathered} max \quad \rho \\ subject \quad to \quad y_i \sum_j w_j P_j \left[\Phi(x_i)\right] \geq \rho \\ for \quad i = 1...n \\ \|w\|_2 = 1 \end{gathered}$$

SVM과 Boosting간의 관계는 Φ(x), 즉 kernel 함수가 hypothesis h(x)와 비슷한 꼴이라는 점에서 더 명확하게 해석해볼 수 있다.

$$\Phi : x \mapsto [h_1(x), ..., h_N(x)]^T$$

그래서

$$k(x,y) = (\Phi(y) * \Phi(y)) = \sum_j h_j(x) h_j(y)$$

이러한 식이 만족이 될 수 있다.(한마디로 정리하면, 그냥 h(x)라는 여러 weak learner 모델들을 커널이라는 형태로 해석할 수 있다는 것이다.)

참고

Unsupervised Learning

Intro

Unsupervised learning은 input만 주어지고 그에 따른 output이 없는 데이터를 가지고 수행하는 학습이다.

Clustering, Density estimation, 그리고 Data description 등의 task가 이에 속한다.

지금까지 계속 다루고 있는 kernel trick들 역시 unsupervised learning에서 유용하게 쓰일 수 있다.

이 장에서는 고차원으로 매핑된 공간에서 수행하는 PCA인 Kernel PCA,
그리고 anomaly detection에서 최근 쓰이는 One-Class SVM을 통해 비지도 학습에서 커널 트릭을 공부한다.

I. PCA

1. PCA?

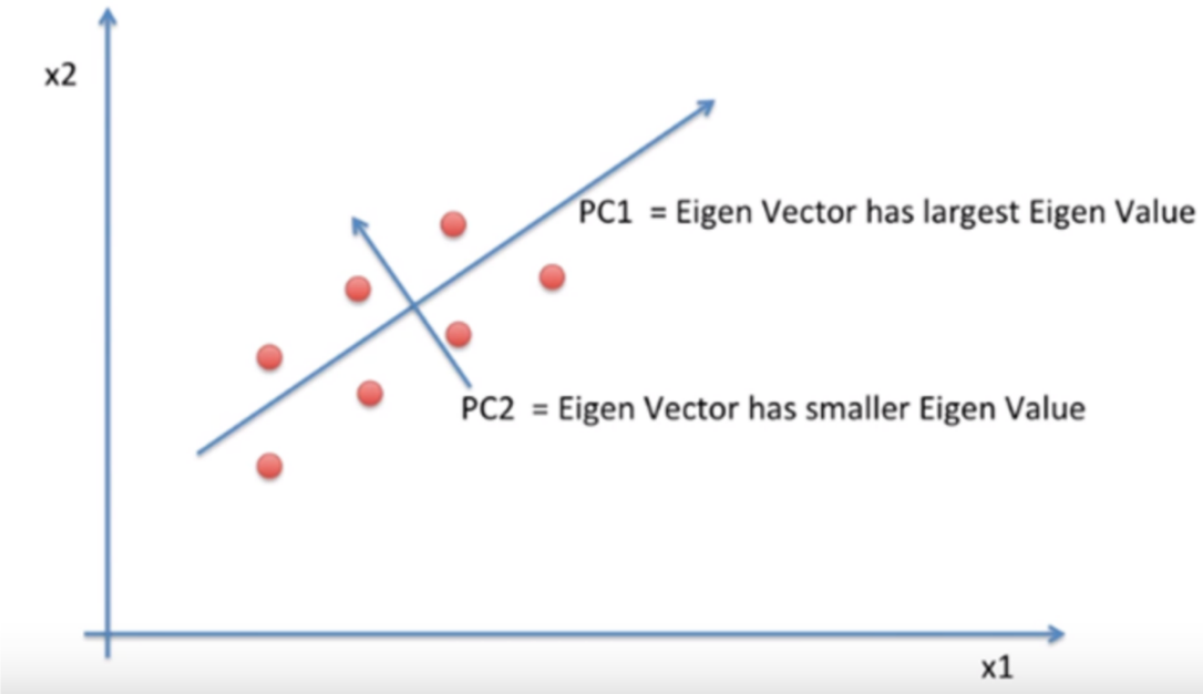
PCA를 조금만 복습해볼까?

PCA는 데이터의 분산을 가장 잘 capture하는 새로운 축으로 데이터를 사영하여 차원을 축소하는 기법이다.

이 때 계산된 새로운 축을 Principal Component, PC라고 하는데, 이 PC는 공분산행렬의 고유값 벡터를 계산해 얻어낼 수 있다.

PCA를 통한 차원축소를 수행함으로써 우리는 데이터의 구조에 대한 설명력을 가짐과 동시에 그 과정에서 정보의 손실을 최소화한다.

그러나 기본적으로 PCA는 linear한 algorithm이기 때문에 nonlinear한 구조를 갖는 데이터에 대해서 좋은 결과를 주기 어렵다. 이 때 Kernel PCA를 통해 nonlinear data를 고차원 공간으로 mapping하여 주성분을 뽑아낼 수 있다.



2. Kernel PCA?

Kernel PCA를 수행하기 위해 가장 먼저 해야할 것은
$\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^N$를 feature space $\mathcal{F}$로 mapping 하는 작업이다. 이후 이로부터 공분산 행렬을 계산한다.

PCA를 수행하기 위해 data는 모두 standardized 되어 있는 상태임을 기억하자(단위가 조정되지 않으면 분산이 왜곡되기 때문).

두 관측벡터 x_j와 x_k의 공분산은 $\sigma_{jk} = \frac{1}{2}\sum_{i=1}^{n}(x_j^{(i)} - \mu_j)(x_k^{(i)} - \mu_k)$로 계산되고,
이 때 $\mu_j = \mu_k = 0$이다.

따라서 위 식을 다시 쓰면 $\sigma_{jk} = \frac{1}{2}\sum_{i=1}^{n} x_j^{(i)} x_k^{(i)}$가 된다.

이로부터 일반화된 공분산 행렬을 계산하면 $C = \frac{1}{n}\sum_{j=1}^{n}(\mathbf{x}_j)(\mathbf{x}_j)^\top$가 되고 이제 데이터를 feature space로 mapping 했다고 하면

$$C = \frac{1}{n}\sum_{j=1}^{n}(\Phi(\mathbf{x}_j))(\Phi(\mathbf{x}_j))^\top$$

라는 공분산 행렬이 최종적으로 완성된다.

PC를 찾기 위한 Eigenvector를 계산하기 위해

$$find\ \lambda > 0, \boldsymbol{V} \neq 0\ with$$
$$\lambda \boldsymbol{V} = C\boldsymbol{V} = \frac{1}{n}\sum_{j=1}^{n}(\Phi(\mathbf{x}_j) \cdot \boldsymbol{V})\Phi(\mathbf{x}_j)$$

와 같은 Eigenvalue problem을 적용하면 되는데,

이 때 기억해야 할 것은 0이 아닌 $\boldsymbol{V}$에 대한 모든 solution들은
mapping된 데이터들의 span에 속한다는 점이다. (Eigenvector의 정의를 생각하면 당연하다)

즉,

$$\boldsymbol{V} \in span\{\Phi(\mathbf{x}_1), ..., \Phi(\mathbf{x}_n)\}$$

이기에 이로부터 $\lambda\langle\Phi(\mathbf{x}_k), \boldsymbol{V}\rangle = \langle\Phi(\mathbf{x}_k), C\boldsymbol{V}\rangle$를 induce 할 수 있고, 또한 $\boldsymbol{V}$를

$$There\ exists\ \alpha_i\ s.t.$$
$$\boldsymbol{V} = \sum_{i=1}^{n}\alpha_i\Phi(\mathbf{x}_i)$$

라고 표현할 수도 있다. 이렇게 표현된 $\boldsymbol{V}$를 $\lambda\langle\Phi(\mathbf{x}_k), \boldsymbol{V}\rangle = \langle\Phi(\mathbf{x}_k), C\boldsymbol{V}\rangle$에 대입하면

$$\boldsymbol{K}\lambda\boldsymbol{\alpha} = \boldsymbol{K}^2\boldsymbol{\alpha},$$
$$\therefore \lambda\boldsymbol{\alpha} = \boldsymbol{K}\boldsymbol{\alpha}, \quad where\ K_{ij} = \langle\Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j)\rangle$$

우리는 이로부터 Kernel PCA의 수행이 mapping 함수와 독립이고 오로지 Kernel function에만 dependent 하다는 사실을 발견할 수 있다. 즉, 지금까지와 마찬가지로 Kernel function만 알면 Kernel PCA를 수행할 수 있다.

아직 끝난 게 아닌데 왜냐하면 $\mathcal{F}$ 위에서 Eigenvector의 normalization과 데이터의 centering이 남아있기 때문이다.

$\boldsymbol{N}$개의 Eigenvector가 있고 first non-zero Eigenvector가 p번째부터라고 할 때, normalize 하기 위해선 $\langle \boldsymbol{V}_k, \boldsymbol{V}_k \rangle = 1,\ where\ k = p, ..., \boldsymbol{N}$을 만족하면 된다.

이를 수식으로 정리해보면

$$\langle \sum_{i=1}^{\boldsymbol{N}} \alpha_i^k \Phi(\mathbf{x}_i), \sum_{j=1}^{\boldsymbol{N}} \alpha_j^k \Phi(\mathbf{x}_j) \rangle = 1,$$
$$\sum_i \sum_j \alpha_i^k \alpha_j^k K_{ij} = 1,$$
$$\langle \boldsymbol{\alpha}_k, K\boldsymbol{\alpha}_k \rangle = 1,$$
$$\therefore \lambda_k \langle \boldsymbol{\alpha}_k, \boldsymbol{\alpha}_k \rangle = 1$$

위 제약조건을 만족하는 α_i를 찾으면 normalization condition을 충족할 수 있다.

Centering in feature space 역시 간단하다.

$$\tilde{\Phi}(\mathbf{x}_t) = \Phi(\mathbf{x}_t) - \frac{1}{N}\sum_{l=1}^{N}\Phi(\mathbf{x}_l),$$
$$\tilde{K}_{ij} = \langle \tilde{\Phi}(\mathbf{x}_i), \tilde{\Phi}(\mathbf{x}_j) \rangle = \langle \Phi(\mathbf{x}_i) - \frac{1}{N}\sum_{l=1}^{N}\Phi(\mathbf{x}_l), \Phi(\mathbf{x}_j) - \frac{1}{N}\sum_{k=1}^{N}\Phi(\mathbf{x}_k) \rangle$$
$$= K_{ij} - \frac{1}{N}\sum_k K_{ik} - \frac{1}{N}\sum_l K_{lj} + \frac{1}{N^2}\sum_l\sum_k K_{lk}$$

$$\therefore \tilde{\boldsymbol{K}} = \boldsymbol{K} - \mathbf{1}_N\boldsymbol{K} - \boldsymbol{K}\mathbf{1}_N + \mathbf{1}_N\boldsymbol{K}\mathbf{1}_N, \quad where$$
$$\mathbf{1}_N = \frac{1}{N}\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

kernel matrix K를 위처럼 구한 $\tilde{K}$로 대체하면 된다.

이제 다 끝났다.

$\mathcal{F}$에서 PC를 뽑아내고 싶다면 간단하게 mapping된 $\mathbf{x}$, 즉 $\Phi(\mathbf{x})$를 $\boldsymbol{V}^k$위로 사영(내적)하면 된다.

$$
\begin{aligned}
(\boldsymbol{V}^k \cdot \Phi(\mathbf{x})) &= \sum_{i=1}^{M} \alpha_i^k (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x})) \\
&= \sum_{i=1}^{M} \alpha_i \mathrm{k}(\mathbf{x}_i, \mathbf{x}).
\end{aligned}
$$

기억해야 할 점은 Kernel PCA의 과정에서 Kernel Matrix $\boldsymbol{K}$를 구하는 단계와 마지막에 $\mathcal{F}$에서 PC를 계산하는 단계(이마저도 커널 트릭으로 대체된다)를 제외하면 직접 mapping function을 사용할 필요가 없다는 점이다.

따라서 Kernel PCA도 일반 PCA와 마찬가지로 그냥 Eigenvalue problem을 해결함으로써 PC를 찾아낼 수 있다.

II. One-Class SVM

1. Intro

Unsupervised learning에서 가장 대표적인 task는 density estimation이다.
그러나 이는 종종 수행하기 어려운 작업이기도 하다.

이유 1: 분포가 존재하지 않을 수 있다.

이유 2: 정확한 분포를 추정하는 것은 어렵다.

따라서 대개 작업은 정확한 분포를 추정하기보다 support of a data distribution만 추정한다.

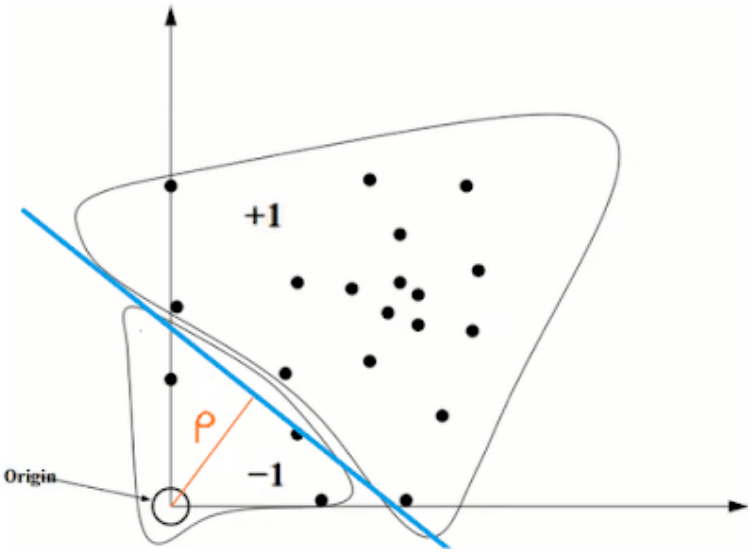
P.S. support of a data distribution:
the closure of the set of possible values of a r.v.s having that distribution

One-Class SVM 역시 이에 초점을 두었다.

2. One-Class SVM

Schölkopf에 의해 제시된 이 방법은
margin의 개념을 차용해 abnormal sample을 normal ones로부터 구분한다.

Data를 feature space $\mathcal{F}$로 사영한 후
normal data들을 원점으로부터 최대한 떨어지도록 하는 hyperplane을 그리는 과정이다.



이 hyperplane (그림에서 하늘색 선)을 그리기 위한 optimization problem은 quadratic program을 따른다.

$$\min_{\mathbf{w}\in F,\boldsymbol{\xi}\in\mathbb{R}^n,\rho\in\mathbb{R}} \quad \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{\nu n}\sum_i \xi_i - \rho$$
$$s.t.\ (\mathbf{w}\cdot\Phi(x_i)) \geq \rho - \xi_i,\ \xi_i \geq 0$$

$\frac{1}{2}\|\mathbf{w}\|^2$는 x 의 변화에 따른 변화가 크지 않도록 (flatness를 유지하도록) 하는 regularization term이다.

ρ는 그림에서 주황색 직선으로, 원점과 hyperplane과의 거리를 의미한다.

이 값을 식에서 빼줌으로써 최소화 문제를 만족하는 최적해일 때 ρ가 최대가 되도록 한다.
(원점에서 hyperplane이 최대한 멀어지도록 만든다.)

$\frac{1}{\nu n}\sum_i \xi_i$는 hyperplane 안쪽으로 오분류 된 normal sample에 대한 페널티의 합이다.

위의 optimization term을 풀어 hyperplane을 찾았다면, 아래 식을 통해 normal과 abnormal을 구분한다.

$$f(\mathbf{x}) = sign(\sum_i \alpha_i \mathrm{k}(\mathbf{x}_i, \mathbf{x}) - \rho))$$

이 때 α는 당연히 라그랑주 승수법을 사용해 찾아낸다.

- Primal Lagrangian Problem(Minimization)

$$L = \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i - \rho - \sum_{i=1}^{n}\alpha_i(\mathbf{w}\cdot\Phi(x_i) - \rho + \xi_i) - \sum_{i=1}^{n}\beta_i\xi_i$$

by KKT conditions,

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n}\alpha_i\Phi(x_i) = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^{n}\alpha_i\Phi(x_i)$$

$$\frac{\partial L}{\partial \xi_i} = \frac{1}{\nu n} - \alpha_i - \beta_i = 0 \quad \Rightarrow \quad \alpha_i = \frac{1}{\nu n} - \beta_i$$

$$\frac{\partial L}{\partial \rho} = -1 + \sum_{i=1}^{n}\alpha_i = 0 \quad \Rightarrow \quad \sum_{i=1}^{n}\alpha_i = 1$$

- Dual Lagrangian Problem(Maximization) by applying $\mathbf{w} = \sum_{i=1}^{n}\alpha_i\Phi(x_i)$

$$L = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j\Phi(x_i)\Phi(x_j) + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i - \rho - \sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j\Phi(x_i)\Phi(x_j) + \rho\sum_{i=1}^{n}\alpha_i - \sum_{i=1}^{n}\alpha_i\xi_i - \sum_{i=1}^{n}\beta_i\xi_i$$

식을 정리한 후 $\alpha_i = \frac{1}{\nu n} - \beta_i$와 $\frac{1}{\nu n} - \alpha_i - \beta_i = 0$을 이용해 식을 전개하면

$$L = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j\Phi(x_i)\Phi(x_j)$$

이 때 L에 대한 최대화 문제를 $L' = -L$일 때 L'의 최소화 문제로 바꿀 수 있으므로

$$\min_{\boldsymbol{\alpha}} \quad \frac{1}{2}\sum_{ij}\alpha_i\alpha_j \mathrm{k}(x_i, x_j)$$
$$s.t. \quad 0 \leq \alpha_i \leq \frac{1}{\nu n},\ i = 1, ..., n,\ \sum_{i=1}^{n}\alpha_i = 1$$

이로부터 α_i를 구하고 그것을 대입해 최적의 hyperplane을 찾는다.

위 최적화 문제에서 ν 파라미터는 당연히 fraction of error를 조절해준다.

i. ν is an upper bound on the fraction of outliers.

ii. ν is a lower bound on the fraction of SVs.

조금 더 쉽게 설명해보자...

$0 \leq \alpha_i \leq \frac{1}{\nu n}$, 그리고 $\sum_{i=1}^{n}\alpha_i = 1$라는 제약조건이 있다.

α_i의 최대값이 $\frac{1}{\nu n}$이므로 $\sum_{i=1}^{n}\alpha_i = 1$를 만족하려면 최소한 νn개의 α_i가 최대값인 element가 있어야 한다.

- At least νn support vectors exist

$\alpha_i = \frac{1}{\nu n}$이라면, $\alpha_i + \beta_i = \frac{1}{\nu n}$에 따라 $\beta_i = 0$이 된다.
그러나 라그랑주 승수법에 따른 조건($\beta_i\xi_i = 0$)에 따라 둘 모두가 0이 될 수 없기 때문에 (보충) $\xi_i > 0$인, 즉 hyperplane 바깥에 위치하는 support vector가 된다.

따라서 만약 모든 α_i가 $\frac{1}{\nu n}$의 값을 갖는다면, 최대 νn개의 support vector가 hyperplane 바깥에 위치한다.

- At most νn support vectors can be located outside the hyperplane

III. How to Find Best Kernel?

1. Model Selection Technique

Kernel을 사용하는 여러가지 방법론들은 하나의 공통점이 있다.

바로 “어떤 Kernel을 사용하느냐"에 따라 모델 성능이 확연하게 차이난다는 점이다.

*Model Selection Tenique*은 주어진 데이터와 문제 상황에서
가장 적절한 Kernel을 찾아내는 방법을 일컫는다.

세 가지의 model selection tenique이 존재하고,
이들은 모두 일반화된 오차(generalized error)를 예측하는 식으로 작동한다.

generalized error?: unobserved inputs에서 발생하는 오차

1) Bayesian evidence framework

SVM의 작동 방식을 베이즈 추론의 형태로 해석하여 evidence(사전확률)를 최대화 하는 model(kernel)을
찾는 방식,, 이라고 한다.

Relevance Vector Machine (RVM)이라는 이름으로 불리는데,
이 내용만 해도 하나의 논문이기 때문에 관련하여 링크를 남긴다.

Relevance Vector Machine

Summary! RVM은 Bayesian SVM입니다! 특징: evidence approximation 과정에서 SVM보다 더 sparse해진다는 특징이 있습니다. 장점: 더 sparse 한데도 성능은 꽤 좋습니다. hyper-parameter tuning은 자동적으로 결정될

https://euphoria0-0.github.io/posts/Relevance-Vector-Machine/

2) PAC(Probabily Approximately Correct)

PAC는 수학적으로 머신러닝 모델을 분석해주는 방법론으로,
"높은 확률로(probably) 주어진 모델이 가장 작은 에러를 갖는다(Approximately Correct)"
는 분석을 가능케 한다.

이에 대한 설명에 앞서 머신러닝에서 '좋은 모델'이란 무엇인지를
computational learning view에서 해석할 필요가 있다.

머신러닝의 궁극적인 목표는 결국 주어진 데이터로부터 원하는 결과를 정확히 예측하는 function을 찾는 것이다.

몇 가지 용어를 정리하고 넘어가도록 하자.

- Instance X: 모든 input을 담고 있는 공간이다
- Target concept c: Instance space의 subset으로 정의되며, 주어진 데이터가 어떤 값을 갖는지 판단하는 함수이다.
- Hypothesis space H: c와 최대한 비슷한 approximated function h가 속하는 function space이다.
- Training data D: Instance space는 절대로 전부 주어지지 않기 때문에 그 중 일부인 training data만으로 c를 추정한다(h).

머신러닝에서 우리의 모델은 전체 데이터셋 $x \in X$에 대해 가장 target concept $c(x)$와 비슷한 hypothesis $h(x)$를 찾아내야 한다.

그러나 실제로는 training sample $x \in D$만 관측할 수 있으므로 우리는 training error를 기준으로 $h(x)$를 평가한다.

그렇다면 과연 D에서 가장 작은 error를 보이는 모델이 전체 X에 대해서도 가장 작은 error를 보장할까?

위 질문에서 후자를 true error라고 명명하면 두 가지 타입의 error는 아래처럼 정의된다.

$$error_{true}(h) := Pr_{x \in X}[c(x) \neq h(x)]$$
$$error_{train}(h) := Pr_{x \in D}[c(x) \neq h(x)]$$

당연히 true error는 무조건 추정된 값일 수밖에 없다.

그러나 우리가 만약 test data를 i.i.d하게 뽑아낸다면 test error는 true error의 unbiased estimator가 된다.

따라서 우리는 train error를 minimize하는 방식으로 모델을 학습하며, 이 과정을 empirical risk minimization이라고 한다.

이러한 개념으로부터 Overfitting을 다시 정의할 수 있다.

만약 다음과 같은 $h' \in H$가 존재할 때 h는 train data에 대해 overfitting 되었다고 정의한다.

$$error_{train}(h) < error_{train}(h') \; and \; error_{true}(h) > error_{true}(h')$$

즉, train error는 h보다 큰데 true error는 그보다 작은 어떤 가설 h'이 존재하면 가설 h는 overfitted된 것이다.

이제 다시 PAC에 대한 설명으로 넘어가보자.

PAC는 generalization error를 bounded된 형태로 표현해준다. 이 때 bound는 4가지 요소로 구성된다.

- Training data의 number of samples m
- Train error와 true error 사이의 gap ε, $where \quad error_{true} - error_{train} \leq \varepsilon$
- Hypothesis space H의 complexity $|H|$.
- Confidence of the relation: at least $1 - \sigma$

Finite hypothesis space에 대해서 PAC bound는 다음과 같이 주어진다.

$$Pr[error_{true}(h) \leq error_{train} + \varepsilon] \leq \|H\| exp(-2m\varepsilon^2)$$

즉, true error와 train error의 gap의 upper bound를 training data의 개수와 모델의 복잡도, 그리고 error와 그 confidence ε과 σ로 표현하는 Bound인 셈이다.

이 bound를 우리는 어떻게 해석해야 할까?

기억해야 할 사실은 ε이 generalization과 overfitting에 대한 의미를 함축하고 있다는 점이다.

ε은 그 정의 자체로 training error와 generalization error의 차이이다.

또한 ε이 커질수록 위 overfitting의 정의에서 h'이 존재할 확률이 커진다.

따라서 PAC bound로부터 얻을 수 있는 intuition은 다음과 같다.

- m (=# of training data)이 커질수록 ε이 작아져 모델은 더욱 generalized 되고 overfitting의 위험이 줄어든다.
- $|H|$가 클수록 generalization은 어려워지고 overfitting의 위험도 커진다.
- 따라서 generalization error를 minimize하려면 train error를 줄이거나 training sample을 더 많이 가져가거나 모델의 complexity를 줄여야 한다.

즉, 우리가 직관적으로 알고 있던 모델의 성능을 높이는 방법이 수식으로 표현된 셈이다.

결국 PAC에 기반한 model selection technique은 generalization error의 bound를 minimize하는 kernel을 찾는 과정이다.

3) Cross Validation

다 아는 방식이다.

Cross validation을 통해 가장 작은 averaged error가 최소인 kernel function을 선택하면 된다.

위 세 가지 방법론 중 가장 자주 사용되는 방식은 교차검증 방식이다.

그러나 명확한 단점이 있는데 바로 **Computationally Expensive** 하다는 점이다.

그래서 SVM 기반의 모델에서는 LOOCV가 자주 쓰인다. 그런데 LOOCV는 K-fold CV보다 훨씬 computationally expensive하지 않나?

SVM에서 모델의 구축에 필요한 건 support vector들 뿐이다. 그런데 LOOCV는 딱 한 개의 observation만을 교체로 바꿔가며 error를 average out 하는 방식이다.

따라서 **observation 하나의 유무가 support vector의 구성에 영향을 미치지 않는다**는 가정하에 LOOCV를 진행하면, 모든 경우에 같은 값의 error가 도출되기 때문에 한 번의 계산만 해도 괜찮다!

참고

Support Vector Machine (SVM)

KFD

Unsupervised Learning

⬇️ Full text down below ⬇️

https://s3.us-west-2.amazonaws.com/secure.notion-static.com/6ba74f3a-0755-4553-bbe8-b3879e37adb9/An_Introduction_to_kernel-based_learning_algorithms.pdf?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Content-Sha256=UNSIGNED-PAYLOAD&X-Amz-Credential=AKIAT73L2G45EIPT3X45%2F20220109%2Fus-west-2%2Fs3%2Faws4_request&X-Amz-Date=20220109T134055Z&X-Amz-Expires=86400&X-Amz-Signature=eb4e3cebc07bd59320e2f0c53199ce3030b8d9856b83a0fe0d61ed9090053fe4&X-Amz-SignedHeaders=host&response-content-disposition=filename%20%3D%22An%2520Introduction%2520to%2520kernel-based%2520learning%2520algorithms.pdf%22&x-id=GetObject